Combining Faceted Search and Query Languages for the Semantic Web

Sébastien Ferré, Alice Hermann, Mireille Ducassé

Combining Faceted Search and Query Languages for the Semantic Web

Sébastien Ferré[1], Alice Hermann[2], and Mireille Ducassé[2]

[1] IRISA/Université de Rennes 1, Campus de Beaulieu, 35042 Rennes cedex, France
ferre@irisa.fr
[2] IRISA/INSA de Rennes, Campus de Beaulieu, 35708 Rennes cedex 7, France
{alice.hermann,ducasse}@irisa.fr

Abstract Faceted search and querying are the two main paradigms to search the Semantic Web. Querying languages, such as SPARQL, offer expressive means for searching knowledge bases, but they are difficult to use. Query assistants help users to write well-formed queries, but they do not prevent empty results. Faceted search supports exploratory search, i.e., guided navigation that returns rich feedbacks to users, and prevents them to fall in dead-ends (empty results). However, faceted search systems do not offer the same expressiveness as query languages. We introduce *semantic faceted search*, the combination of an expressive query language and faceted search to reconcile the two paradigms. The query language is basically SPARQL, but with a syntax that better fits in a faceted search interface. A prototype, Camelis 2, has been implemented, and a usability evaluation demonstrated that semantic faceted search retains the ease-of-use of faceted search, and enables users to build complex queries with little training.

1 Introduction

With the growing amount of available resources in the Semantic Web (SW), it is a key issue to provide an easy and effective access to them, not only to specialists, but also to casual users. The challenge is not only to allow users to retrieve particular resources (e.g., flights), but to support them in the exploration of a knowledge base (e.g., which are the destinations? Which are the most frequent flights? With which companies and at which price?). We call the first mode *retrieval search*, and, following Marchionini [Mar06], the second mode *exploratory search*. Exploratory search is often associated to *faceted search* [HEE+02,ST09], but it is also at the core of Logical Information Systems [Fer09], and Dynamic Taxonomies [Sac00]. Exploratory search allows users to find information without *a priori* knowledge about either the data or its schema. Faceted search works by suggesting restriction values, i.e., selectors for subsets of the current selection of items. Restriction values are organized into facets, and only those that share items with the current selection are suggested. This has the advantage to remove the need to write queries, and to prevent dead-end queries, i.e., queries with no answer. Therefore, faceted search is *easy* and *safe*: *easy* because users only have

to choose among the suggested restriction values, and *safe* because, whatever the choice made by users, the resulting selection is not empty. The selections that can be reached by navigation correspond to queries that are generally limited to conjunctions of restriction values, possibly with restricted negation and disjunction. This is far from the expressiveness of query languages for the semantic web, such as SPARQL[1]. SlashFacet [HvOH06] and BrowseRDF [ODD06] are faceted search systems for RDF data that extend the expressiveness of reachable queries, but still to a small fragment of SPARQL. For instance, both of them allow for neither cycles in graph patterns, nor unions of graph patterns (disjunction).

Querying languages for the semantic web, such as SPARQL [AG08], OWL-QL [FHH04], or SPARQL-DL [SP07], are quite expressive but are difficult to use, even for specialists. They do not return enough feedback to offer exploratory search, and nothing prevents users to write a query that has no answer. Indeed, even if users have a perfect knowledge of the syntax and semantics of the query language, they may be ignorant about the data schema, i.e., the *ontology*. If they also master the ontology or if they use a query assistant (e.g., Protégé[2]) or an auto-completion system (e.g., Ginseng [BKK05]), the query will be syntactically correct and semantically consistent w.r.t. the ontology but it can still produce no answer.

The contribution of this paper is to extend faceted search to the Semantic Web, so as to offer an exploratory search that is (1) easy to use, (2) safe, and (3) expressive. Ease-of-use and safeness are retained from existing faceted search systems by keeping their general principles, as well as the visual aspect of their interface. Expressiveness is obtained by representing the current selection by a *query* rather than by a set of items, and by representing navigation links by *query transformations* rather than by set operations (e.g., intersection). In this way, the expressiveness of faceted search is determined by the expressiveness of the query language, rather than by the combinatorics of user interface controls. In this paper, the query language is based on SPARQL graph patterns, but with a syntax that better fits in a faceted search interface: LISQL.

The use of queries for representing selections in faceted search has other benefits than navigation expressiveness. The current query is an intensional description of the current selection that complements its extensional description (listing of items). It informs users in a precise and concise way about their exact position in the navigation space. It can easily be copied and pasted, stored and retrieved later. Finally, it allows expert users to modify the query by hand at any stage of the navigation process, without loosing the ability to proceed by navigation.

The paper is organized as follows. Section 2 presents *semantic faceted search*, and illustrates it with our prototype implementation Camelis 2. Section 3 reports about a user study that demonstrates the usability of our approach. Our approach is also compared in Section 4 to other work in faceted search for the Semantic Web. Section 5 concludes this paper.

[1] see `http://www.w3.org/TR/rdf-sparql-query/`
[2] See `http://protege.stanford.edu/`

2 Semantic Faceted Search

The principle of our approach, *Semantic Faceted Search* (SFS), is to reconcile querying and navigation. Navigation can be defined as moving from place to place through navigation links. In faceted search, a navigation place is a set of items, and a navigation link is the choice of a restriction value. In SFS, a navigation place is defined by a query, whose answers form the current set of items; and a navigation link is defined as a query transformation. The set of possible query transformations is designed to make it possible to build arbitrary queries. However, the set of navigation links is restricted to those query transformations that do not lead to empty results, like in standard faceted search. In the following, SFS is illustrated on genealogical datasets converted from GED files[3].

2.1 Queries and Query Transformations

The reference query language for the Semantic Web is SPARQL. While its semantics is adequate to our needs, we find that its syntax is not best-suited to SFS. First, it makes it difficult to define query transformations, because it is not regular and compositional enough. Second, it tends to be verbose, and exhibits relational algebra operators, and a number of symbols that are alien to most people: e.g., UNION, `&&`.

We propose an alternative syntax, LISQL, for a large fragment of SPARQL. This fragment corresponds to unary queries, i.e., queries with only one variable in the SELECT clause. This restriction is not due to LISQL, but to the very definition of faceted search, where a navigation place is a set and not a relation. For reasons of space, we present LISQL and its query transformations through examples only. Full definitions can be found in a research report [FHD11].

As an illustrating example covering all aspects of LISQL, we consider the task of retrieving, in the genealogy of George Washington, "every person that was born in 1601 or 1649 at some place in England, and some child of which was not born at the same place". In SPARQL, this query can be expressed as follows.

```
SELECT DISTINCT ?p
WHERE {
    ?p a person.
    ?p birth ?b.
        ?b year ?y FILTER (?y=1601 || ?y=1649).
        ?b place ?X. ?X in England.
    ?c father ?p.
        ?c birth ?bc.
            ?bc place ?pc FILTER ?pc != ?X }
```

The same query can be expressed in LISQL as:

[3] `http://jay.askren.net/Projects/SemWeb/`

```
a person and birth : (year : (1601 or 1649) and place : (?X
and in England)) and father of birth : place : not ?X.
```

A LISQL query is a LISQL expression, where some subexpression, the *query focus*, is underlined. LISQL expressions denote sets of items (RDF resources), and can be coordinated by Boolean operators that correspond to set operations: `and` for intersection, `or` for union, and `not` for complement. Atomic expressions can be individuals (e.g., `1601`, `England`) that denote themselves as singleton sets, or classes (e.g., `person`) that denote their set of instances. The subexpression `year : (1601 or 1649)` is a *restriction*, made of the property `year` and of the subexpression, `(1601 or 1649)`, and here denotes the set of events whose year is 1601 or 1649. The keyword (`of`) is used instead of (`:`) for the inverse reading of a property. The variable `?X` is here used to refer to the birthplace of the father, whatever it is. A LISQL expression translates to a SPARQL graph pattern, using a variable for each entity. The *query focus* determines which of those variables is put in the SELECT clause. Therefore, the query `a woman and father : ?` denotes the women's fathers, and is equivalent to `father of a woman`. A focus is never ambiguous; however, two different foci can be equivalent, i.e., put the same variable in the SELECT clause, if they are conjunctively coordinated or associated to a same LISQL variable. LISQL variables therefore allow for cycles in the graph patterns.

We present the different kinds of query transformations through a possible scenario for building the above query. Table 1 gives for each step the query transformations, and the resulting intermediate query. Most transformations apply to the query focus, letting the rest of the query unchanged. Transformation *Reset* resets the whole query to the most general query `?`. Transformation *And* appends a given subexpression to the focus, connecting the two with `and` (`?` is a neutral element for `and`). When the focus is on the whole expression, this transformation corresponds to faceted search selection. Transformation *Focus on* moves the focus on a given subexpression. Transformation *Cross* is an abbreviation for a common navigation idiom: the *And* of a restriction followed by a *Focus on* the subexpression of the restriction. Transformation *Name* does the same as a *And*, but for a freshly generated variable. The *And* of the same variable at a later stage allows for the formation of cycles. Transformation *Or* introduces an alternative to the focus with the connector `or`. Transformation *Not* applies the connector `not` to the focus.

Only transformation *And* requires a LISQL expression to be passed. A set of expressions is suggested to the user so as to avoid dead-ends, and to allow for arbitrarily complex queries. It is sufficient to suggest individuals, variables already in the query, classes, and unqualified restrictions (e.g., `father of ?`) [FHD11].

2.2 Faceted User Interface and Interaction

SFS has been implemented as a prototype, Camelis 2[4]. Figure 1 shows a screenshot of Camelis 2. From top to bottom, and from left to right, it is composed of

[4] downloadable at `http://www.irisa.fr/LIS/ferre/camelis/camelis2.html`

0	*Reset* ?
1	*And* a person a person
2	*And* birth : year : 1601 a person and birth : year : 1601
3	*Focus on* year : 1601 a person and birth : year : 1601
4	*Cross* place : ? + *Name* a person and birth : (year : 1601 and place : ?X)
5	*And* in England a person and birth : (year : 1601 and place : (?X and in England))
6	*Focus on* 1601 + *Or* a person and birth : (year : (1601 or ?) and place : (?X and in England))
7	*And* 1649 a person and birth : (year : (1601 or 1649) and place : (?X and in England))
8	*Focus on* a person + *Cross* father of birth : place : ? a person and birth : (year : (1601 or 1649) and place : (?X and in England)) and father of birth : place : ?
9	*Not* a person and birth : (year : (1601 or 1649) and place : (?X and in England)) and father of birth : place : not ?
10	*And* ?X + *Focus on* a person a person and birth : (year : (1601 or 1649) and place : (?X and in England)) and father of birth : place : not ?X

Table1. A navigation scenario in Camelis 2 on the genealogy of George Washington.

a menu bar (M), a toolbar (T), a query box (Q), query controls (QC), feature controls (FC), an answer list or extension box (E), a facet hierarchy (F), and a set of value boxes (V). A query engine can be derived from Camelis 2 by retaining only the components Q and E. A standard faceted search system can be derived by retaining only the components E, F, and V.

Navigation links, i.e., suggested query transformations, are available on all components. Whenever a navigation control is triggered, the corresponding query transformation is applied, and components (Q,E,F,V) are refreshed accordingly. The toolbar (T) has a button for *Reset*. The query box (Q) is clickable for setting the focus on any subexpression. Query controls (QC) provide buttons for *Name*, *Or*, *Not* (and a few others). Every element of components (E,F,V) can be used as an argument for *And*, with the guarantee that the resulting query does have answers; *And* is replaced by *Cross* for unqualified restrictions. The contents of components (E,F,V) play the role of restriction values in standard faceted search, and are here dispatched in the three components according to their type. The facet hierarchy (F) contains variables of the current query (e.g., ?X, classes (e.g.,

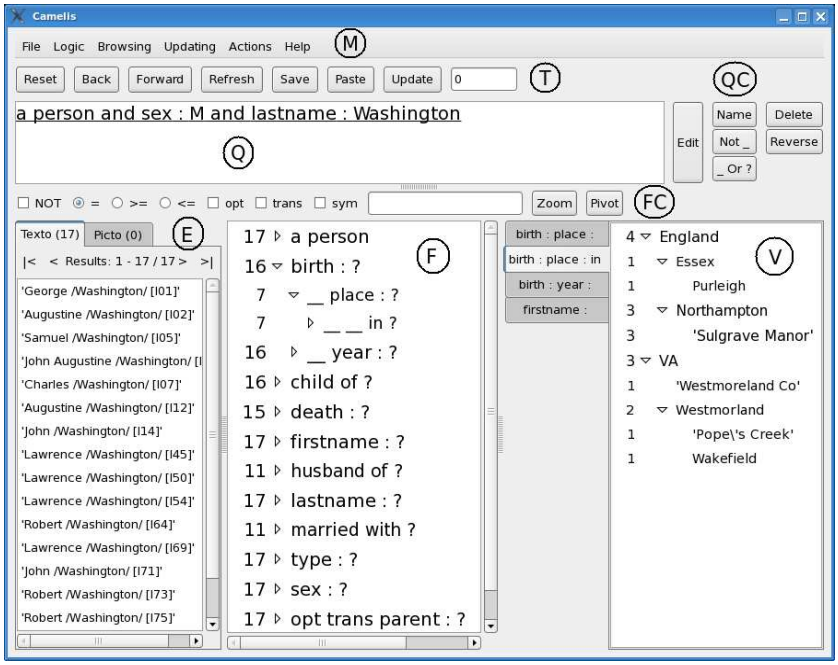

Figure1. A screenshot of the user interface of Camelis 2. It shows the selection of male persons whose lastname is Washington.

`a person`), and *unqualified* restrictions (e.g., `father of ?`, `birth : year : ?`). Value boxes (V) contain *qualified* restrictions with individuals as subexpressions (e.g., `father of 'George Washington'`, `birth : year : 1601`), grouped by property path. The extension box (E) contains individuals (e.g., `England`). The hierarchical organization of facets in (F) is based on RDFS class and property hierarchies. A value box (V) is hierarchically organized according to the last property of its property path, if it is transitive (e.g, `in`).

3 Usability Evaluation

This section reports on the evaluation of Semantic Faceted Search in terms of usability[5]. we have measured the ability of users to answer questions of various complexities, as well as their response times. Results are strongly positive and demonstrate that semantic faceted search offers expressiveness and ease-of-use at the same time.

Methodology. The subjects consisted of 20 graduate students in computer science. They had prior knowledge of relational databases but neither of Camelis 2,

[5] Details can be found on `http://www.irisa.fr/LIS/alice.hermann/camelis2.html`

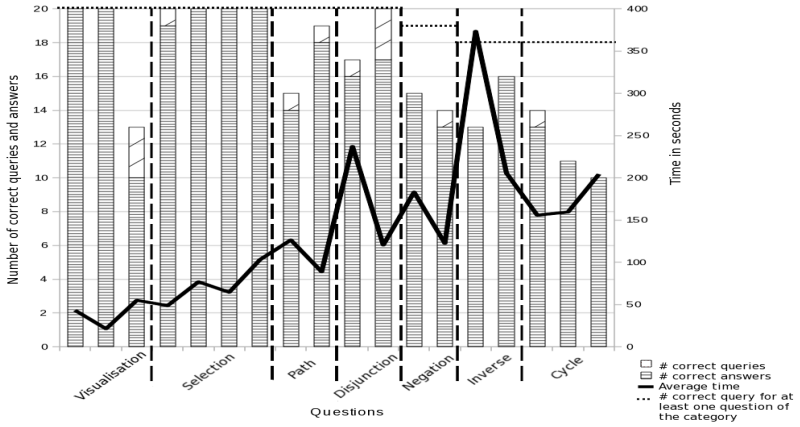

Figure2. Average time and number of correct queries and answers for each question

nor of faceted search, nor of semantic web. None was familiar with the dataset used in the evaluation. The evaluation was conducted in three phases. First, the subjects learned how to use Camelis 2 through a 20min tutorial, and had 10 more minutes for free use and questions. Second, subjects were asked to answer a set of questions, using Camelis 2. We recorded their answers, the queries they built, and the time spent on each question.

The test was composed of 18 questions, with smoothly increasing difficulty. The questions can be grouped in 7 categories: the first 2 categories are covered by standard faceted search, while the 5 other categories are not in general. The first category, *Visualization*, did not require the creation of a query. The exploration of the facet hierarchy was sufficient: e.g., "How many men are there?". In the second category, *Selection*, we asked to count or list items that have a particular feature: e.g., "How many women are named Mary?". In the third category, *Path*, subjects had to follow a path of properties: e.g., "Which man is married with a woman born in 1708?". The fourth category, *Disjunction*, required to use disjunction: e.g., "Which women have for mother Jane Butler or Mary Ball?". The fifth category, *Negation*, required to use negation: e.g., "How many women have a mother whose death's place is not Warner Hall?". The sixth category, *Inverse*, required to use the inverse of a property: e.g., "Who was born in the same place as Robert Washington?". In the seventh category, *Cycle*, required the use of variables: e.g., "How many persons have the same firstname as one of their parent?".

Results. Figure 2 shows the number of correct queries and answers, the average time spent on each question and the number of participants who had a correct query for at least one question of each category. For example, in category "Visualization", the first two questions had 20 correct answers and queries; the third question had 10 correct answers and 13 correct queries; all the 20 participants had a correct query for at least one question of the category; the average re-

sponse times were respectively 43, 21, and 55 seconds. The difference between the number of correct queries and correct answers is explained by the fact that some subjects forgot to set the focus on the whole query after building the query.

All subjects but one had correct answers to more than half of the questions. Half of the subjects had the correct answers to at least 15 questions out of 18. Two subjects answered correctly to 17 questions, their unique error was on a disjunction question for one and on a negation question for the other. All subjects had the correct query for at least 11 questions. For each question, there is at least 50 percent of success. The subjects spent an average time of 40 minutes on the test, the quickest one spent 21 minutes and the slowest one 58 minutes.

The first 2 categories corresponding to standard faceted search, visualization and selection, had a high success rate (between 94 and 100) except for the third question. The most likely explanation for the latter is that the previous question was so simple (`a man`) that subjects forgot to reset the query between the questions 2 and 3. All questions of the first two categories were answered in less than 1 minute and 43 seconds on average. Those results indicate that the more complex user interface of semantic faceted search does not entail a loss of usability compared to standard faceted search for the same tasks.

For other categories, all subjects but two managed to answer correctly at least one question of each category. Within each category, we observed that response times decreased, except for the *Cycle* category. At the same time, for *Path*, *Disjunction* and *Inverse*, the number of correct answers and queries increased. Those results suggest a quick learning process of the subjects. The decrease in category *Negation* is explained by a design flaw in the interface. For category *Cycle*, we conjecture some lassitude at the end of the test. Nevertheless, all but two subjects answered correctly to at least one of *Cycle* questions. The peak of response time in category *Inverse* is explained by the lack of inverse property examples in the tutorial. It is noticeable that subjects, nevertheless, managed to solve the *Inverse* questions with a reasonable success rate, and a decreasing response time.

4 Related Work

As faceted search is becoming widespread, a number of proposals have been made to apply it on the Semantic Web (SW). They all have in common to assume that data is represented in a SW format, either RDF(S) or OWL. Most of them, such as Ontogator [MHS06], mSpace[6], and Longwell[7], do not claim for a contribution in terms of expressiveness, and contribute either to the design of better interfaces and visualizations, or to methods for the rapid or user-centric configuration of faceted views [SVH07]. Therefore, their contributions are somewhat orthogonal to ours, and could certainly complement ours. Other approaches, such as Slash-Facet [HvOH06], BrowseRDF [ODD06], and SOR [LMZ+07], extend faceted search towards a more expressive navigation.

[6] see `http://mspace.fm/`

[7] see `http://simile.mit.edu/wiki/Longwell`

The most essential ingredient for an expressive and flexible semantic search in RDF graphs is *focus change*. It allows to change the perspective without changing the underlying graph pattern. To the best of our knowledge, no faceted search system offers this in a general way. SlashFacet and SOR have the *crossing* operation that selects the images of the items in the current selection through a property. Crossing includes a focus change, but crossing back a property is not equivalent to a focus change, because it introduces an additional restriction: starting from a query Q and crossing p `:` `?` and then p `of` `?` leads to p `:` p `of` Q instead of Q `and` p `:` `?` (they are not equivalent). Other systems allow to focus on different types of items, but this focus cannot be changed in the course of a search. For example, in a dataset about publications, a choice has to be made between authors and documents.

It is generally considered that the query should be hidden from the interface. In fact, in most faceted search systems, the query *is* displayed as the list of the restriction values users have already selected in the course of their search. This is important so that users do not feel lost, and can easily reverse previous selections. When expressiveness is raised to SPARQL with graph patterns, disjunction, and negation, it becomes necessary to introduce syntax. While, in Camelis 2, the query is simply rendered as a sentence following some grammar, nothing prevents to render syntax through graphical widgets (e.g., lists for conjunction, trees for restrictions, tab panels for disjunction).

Disjunction and negation are either absent or strongly limited in existing approaches. Disjunction is restricted to build sets of values or sets of items, e.g., in SlashFacet. Negation is restricted to restriction values, and also applies to unqualified restrictions (`not father of ?`) in BrowseRDF. No other system allows to form cycles as we do with variables.

5 Conclusion

We have introduced *Semantic Faceted Search* (SFS) as a search paradigm for Semantic Web knowledge bases, in particular RDF graphs. It combines the expressiveness of the SPARQL query language, and the benefits of exploratory search and faceted search. The user interface of semantic faceted search includes the user interface of other faceted search systems, and can be used as such. It adds a query box to tell users where they are in their search, and to allow them to change the focus or to remove query parts. It also adds a few controls for applying some query transformations such as insertion/deletion of disjunction, negation, and variables. We have introduced a new query syntax, LISQL, to better fit with a faceted interface and query transformations. Beside the list of selected items, the user interface has a hierarchy of facets organizing classes and properties by subsumption.

SFS has been implemented as a prototype, Camelis 2. Its usability has been demonstrated through a user study, where, after a short training, all subjects were able to answer simple questions, and most of them were able to answer complex questions involving disjunction, negation, or cycles. This means seman-

tic faceted search retains the ease-of-use of other faceted search systems, and gets close to the expressiveness of query languages such as SPARQL.

Acknowledgments. We would like to thank the 20 students, from the University of Rennes 1 and the INSA engineering school, for their volunteer participation to the usability evaluation.

References

[AG08] R. Angles and C. Gutierrez. The expressive power of SPARQL. In A. P. Sheth *et al*, editor, *Int. Semantic Web Conf.*, LNCS 5318, pages 114–129. Springer, 2008.

[BKK05] A. Bernstein, E. Kaufmann, and C. Kaiser. Querying the semantic web with Ginseng: A guided input natural language search engine. In *Work. Information Technology and Systems (WITS)*, 2005.

[Fer09] S. Ferré. Camelis: a logical information system to organize and browse a collection of documents. *Int. J. General Systems*, 38(4), 2009.

[FHD11] Sébastien Ferré, Alice Hermann, and Mireille Ducassé. Semantic faceted search: Safe and expressive navigation in rdf graphs. Research report, 2011.

[FHH04] R. Fikes, P. J. Hayes, and I. Horrocks. OWL-QL - a language for deductive query answering on the semantic web. *J. Web Semantic*, 2(1):19–29, 2004.

[HEE+02] M. Hearst, A. Elliott, J. English, R. Sinha, K. Swearingen, and K.-P. Yee. Finding the flow in web site search. *Communications of the ACM*, 45(9):42–49, 2002.

[HvOH06] M. Hildebrand, J. van Ossenbruggen, and L. Hardman. /facet: A browser for heterogeneous semantic web repositories. In I. Cruz *et al*, editor, *Int. Semantic Web Conf.*, LNCS 4273, pages 272–285. Springer, 2006.

[LMZ+07] Jing Lu, Li Ma, Lei Zhang, J.S. Brunner, Chen Wang, Yue Pan, and Yong Yu. SOR: A practical system for ontology storage, reasoning and search (demo). In *Int. Conf. Very Large Databases (VLDB)*, VLDB Endowment, pages 1402–1405. ACM, 2007.

[Mar06] G. Marchionini. Exploratory search: from finding to understanding. *Communications of the ACM*, 49(4):41–46, 2006.

[MHS06] E. Mäkelä, E. Hyvönen, and S. Saarela. Ontogator - a semantic view-based search engine service for web applications. In I. F. Cruz *et al.*, editor, *Int. Semantic Web Conf.*, LNCS 4273, pages 847–860. Springer, 2006.

[ODD06] E. Oren, R. Delbru, and S. Decker. Extending faceted navigation to RDF data. In I. Cruz *et al*, editor, *Int. Semantic Web Conf.*, LNCS 4273, pages 559–572. Springer, 2006.

[Sac00] G. M. Sacco. Dynamic taxonomies: A model for large information bases. *IEEE Transactions Knowledge and Data Engineering*, 12(3):468–479, 2000.

[SP07] E. Sirin and B. Parsia. SPARQL-DL: SPARQL query for OWL-DL. In C. Golbreich, A. Kalyanpur, and B. Parsia, editors, *Work. OWL Experiences and Directions (OWLED)*, volume 258. CEUR-WS, 2007.

[ST09] G. M. Sacco and Y. Tzitzikas, editors. *Dynamic taxonomies and faceted search*. The information retrieval series. Springer, 2009.

[SVH07] O. Suominen, K. Viljanen, and E. Hyvönen. User-centric faceted search for semantic portals. In E. Franconi, M. Kifer, and W. May, editors, *Eu. Semantic Web Conf.*, LNCS 4519, pages 356–370. Springer, 2007.